**Kafka Processor using Flink and Configuration File Setup**

This document covers my journey of creating a Kafka data processor using Flink with a twist: making it configurable via a properties file.

**My Initial Java Code(Just reading the topic)**

Here's the initial version of my Java code, which was a straightforward Kafka processor using Flink  
package org.example;  
import org.apache.flink.api.common.serialization.SimpleStringSchema;  
import org.apache.flink.streaming.api.datastream.DataStream;  
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeParseException;  
import java.util.Properties;  
import org.apache.flink.core.fs.FileSystem;  
public class KafkaProcessor {  
 public static void main(String[] args) throws Exception {  
 final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
 // Kafka properties  
 Properties properties = new Properties();  
 properties.setProperty("bootstrap.servers", "localhost:9092");  
 properties.setProperty("group.id", "flink-consumer-group");  
 // Kafka Consumer  
 FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>("INPUT\_TOPIC", new SimpleStringSchema(), properties);  
 DataStream<String> input = env.addSource(consumer);  
 // Process data  
 DataStream<String> processed = input.map(line -> {  
   
 });  
 }  
}

**Adding a Configuration File**

Next step: I decided to move the Kafka broker configurations to a separate properties file. This keeps things clean and configurable. The properties file, named 'config.properties', contains keys for the Kafka broker address and group ID. Here is what it looks like:

Location→ src/main/resources/config.properties

*# config.properties file  
bootstrap.servers=localhost:9092  
group.id=flink-consumer-group*

**The Modified Java Code**

Now, I modified my Java code minimally to read the Kafka configuration from this file. Here's the updated code:

package org.example;  
import org.apache.flink.api.common.serialization.SimpleStringSchema;  
import org.apache.flink.streaming.api.datastream.DataStream;  
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;  
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;  
import java.io.InputStream;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeParseException;  
import java.util.Properties;  
import org.apache.flink.core.fs.FileSystem;  
public class KafkaProcessor {  
 public static void main(String[] args) throws Exception {  
 final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
 // Load Kafka properties from config.properties  
 Properties properties = new Properties();  
 try (InputStream input = KafkaProcessor.class.getClassLoader().getResourceAsStream("config.properties")) {  
 if (input == null) {  
 System.err.println("Sorry, unable to find config.properties");  
 return;  
 }  
 properties.load(input);  
 }  
 // Kafka Consumer setup remains same as before, now using the loaded properties.  
 }  
}

**Final Java Code:**

| package org.example;  import org.apache.flink.api.common.serialization.SimpleStringSchema;  import org.apache.flink.streaming.api.datastream.DataStream;  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;  import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;  import java.time.LocalDate;  import java.time.format.DateTimeFormatter;  import java.time.format.DateTimeParseException;  import java.util.Properties;  import org.apache.flink.core.fs.FileSystem;  public class KafkaProcessor {  public static void main(String[] args) throws Exception {  final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  // Kafka properties  //// Properties properties = new Properties();  ////properties.setProperty("bootstrap.servers", "localhost:9092");  ////properties.setProperty("group.id", "flink-consumer-group");  Properties properties = new Properties();  try (InputStream input = KafkaProcessor.class.getClassLoader().getResourceAsStream("config.properties")) {  if (input == null) {  System.err.println("Sorry, unable to find config.properties");  return;  }  properties.load(input);  }  // Kafka Consumer  FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>("INPUT\_TOPIC", new SimpleStringSchema(), properties);  DataStream<String> input = env.addSource(consumer);  // Process data with added logging and validation  DataStream<String> processed = input.map(line -> {  try {  System.out.println("Received input line: " + line);  String[] parts = line.split(",");  if (parts.length != 3) {  System.err.println("Invalid input format: " + line);  return "INVALID: " + line;  }  String name = parts[0].trim();  String address = parts[1].trim();  LocalDate dob;  try {  dob = LocalDate.parse(parts[2].trim(), DateTimeFormatter.ISO\_LOCAL\_DATE);  } catch (DateTimeParseException e) {  System.err.println("Invalid date format for input: " + line);  return "INVALID\_DATE: " + line;  }  Person person = new Person(name, address, dob);  int age = person.getAge();  System.out.println("Processed person: " + person + ", Age: " + age);  // Categorize based on age  if (age % 2 == 0) {  return "EVEN:" + person.toString();  } else {  return "ODD:" + person.toString();  }  } catch (Exception e) {  System.err.println("Error processing line: " + line);  e.printStackTrace();  return "ERROR: " + line;  }  });  // Separate into even and odd streams  DataStream<String> evenStream = processed.filter(value -> value.startsWith("EVEN:"));  DataStream<String> oddStream = processed.filter(value -> value.startsWith("ODD:"));  // Kafka Producers  FlinkKafkaProducer<String> evenProducer = new FlinkKafkaProducer<>(  "EVEN\_TOPIC", new SimpleStringSchema(), properties);  FlinkKafkaProducer<String> oddProducer = new FlinkKafkaProducer<>(  "ODD\_TOPIC", new SimpleStringSchema(), properties);  // Add sinks with logging  evenStream.map(value -> {  String result = value.substring(5); // Remove "EVEN:" prefix  System.out.println("Sending to EVEN\_TOPIC: " + result);  return result;  }).addSink(evenProducer);  oddStream.map(value -> {  String result = value.substring(4); // Remove "ODD:" prefix  System.out.println("Sending to ODD\_TOPIC: " + result);  return result;  }).addSink(oddProducer);  // Optional: Persist messages to a file with logging  processed.map(value -> {  System.out.println("Writing to file: " + value);  return value;  }).writeAsText("/home/azureuser/flink/output.txt", FileSystem.WriteMode.OVERWRITE).setParallelism(1);  // Execute the Flink job  env.execute("Kafka Processor");  }  } |
| --- |

**Directory Structure:**

**chandan@chandans-MacBook-Pro-2 flink-kafka-project % tree**

**.**

**├── README.md**

**├── dependency-reduced-pom.xml**

**├── pom.xml**

**├── src**

**│ ├── main**

**│ │ ├── java**

**│ │ │ └── org**

**│ │ │ └── example**

**│ │ │ ├── KafkaProcessor.java**

**│ │ │ └── Person.java**

**│ │ └── resources**

**│ └── test**

**│ └── java**

**└── target**

**├── classes**

**│ └── org**

**│ └── example**

**│ ├── KafkaProcessor.class**

**│ └── Person.class**

**├── flink-kafka-project-1.0-SNAPSHOT.jar**

**├── flink-kafka-project-shaded.jar**

**├── generated-sources**

**│ └── annotations**

**├── generated-test-sources**

**│ └── test-annotations**

**├── maven-archiver**

**│ └── pom.properties**

**├── maven-status**

**│ └── maven-compiler-plugin**

**│ ├── compile**

**│ │ └── default-compile**

**│ │ ├── createdFiles.lst**

**│ │ └── inputFiles.lst**

**│ └── testCompile**

**│ └── default-testCompile**

**│ ├── createdFiles.lst**

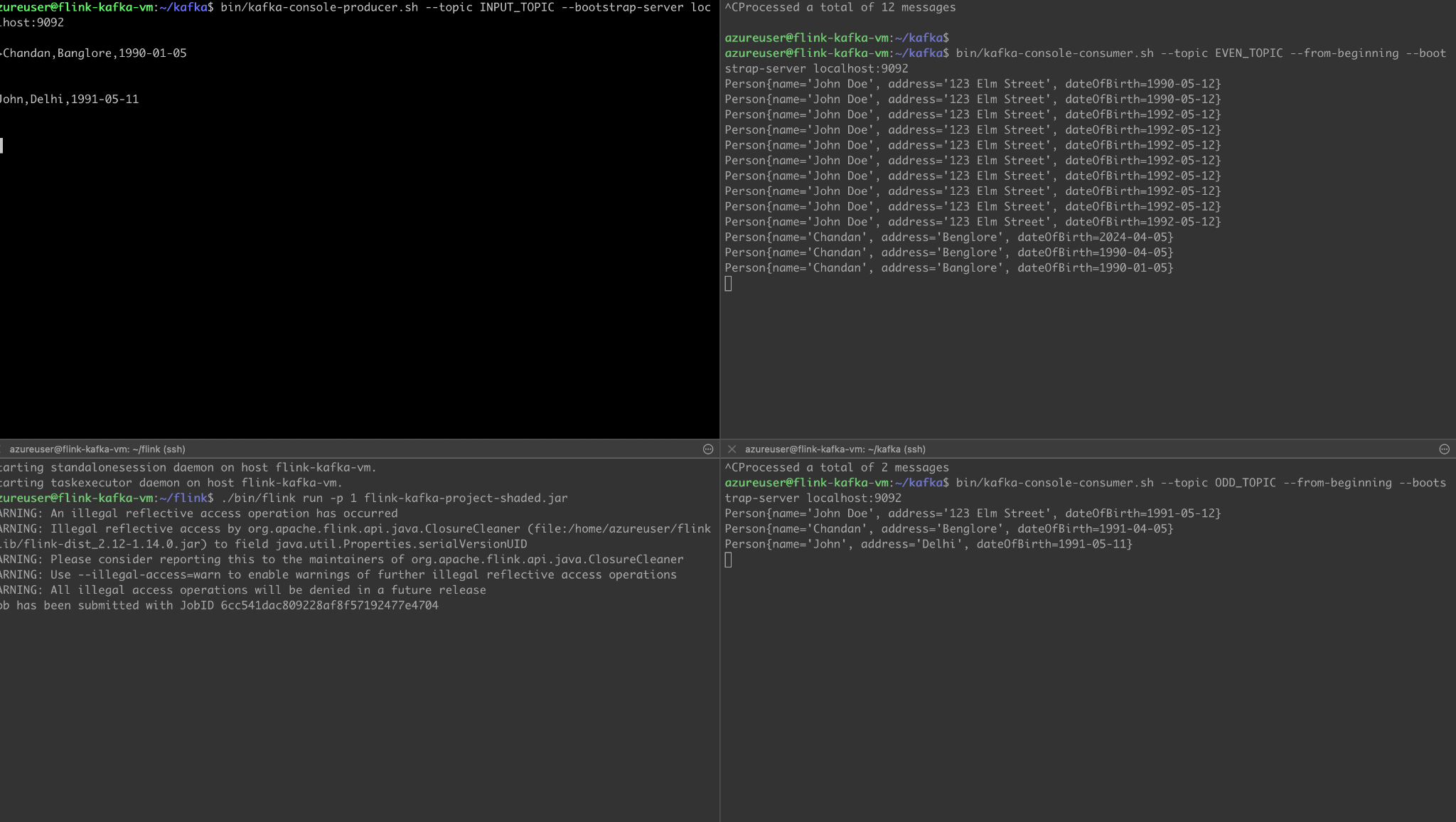
**│ └── inputFiles.lst**

**└── test-classes**

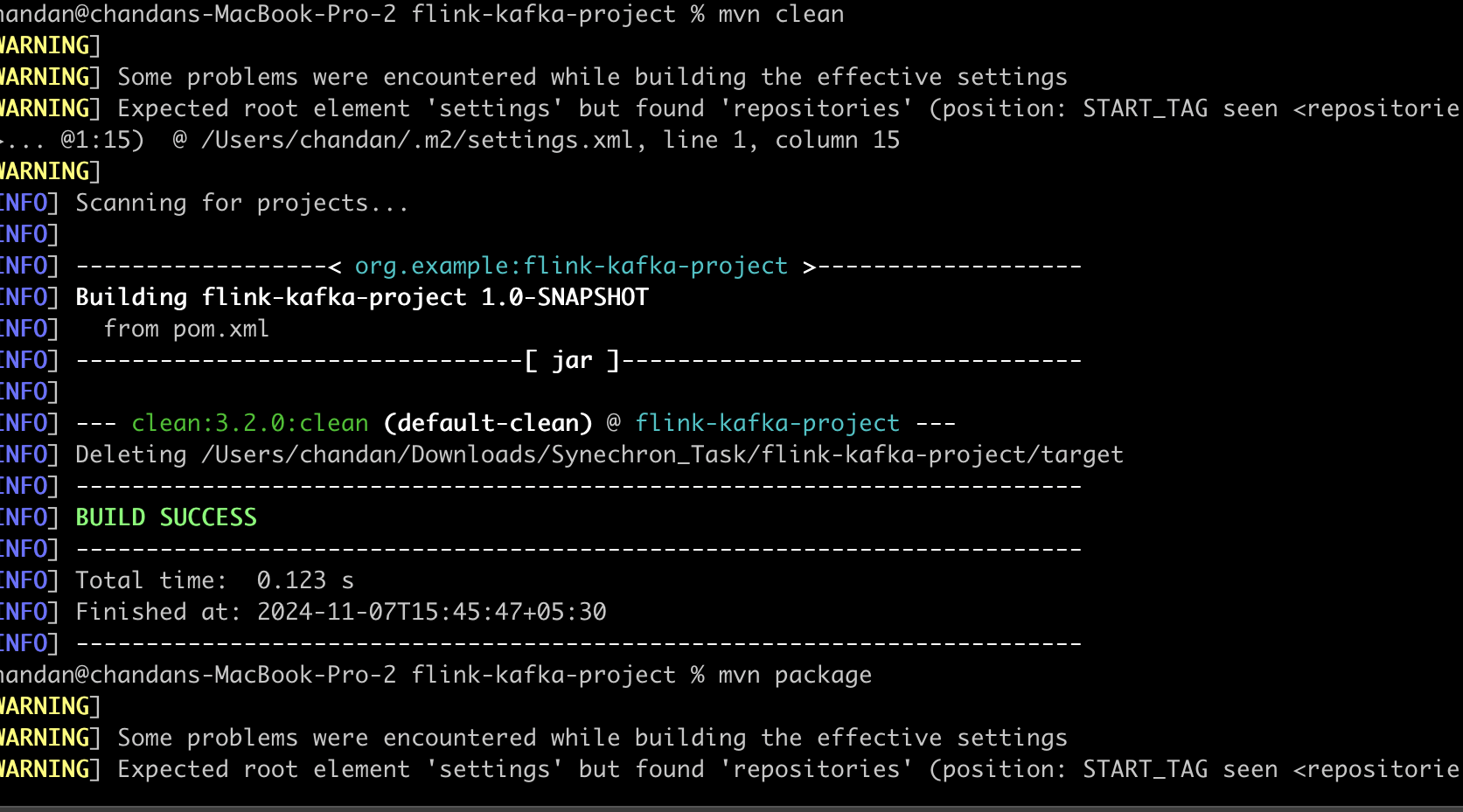
**Screenshots**

Here are a couple of screenshots to show how the code performs after modifications:

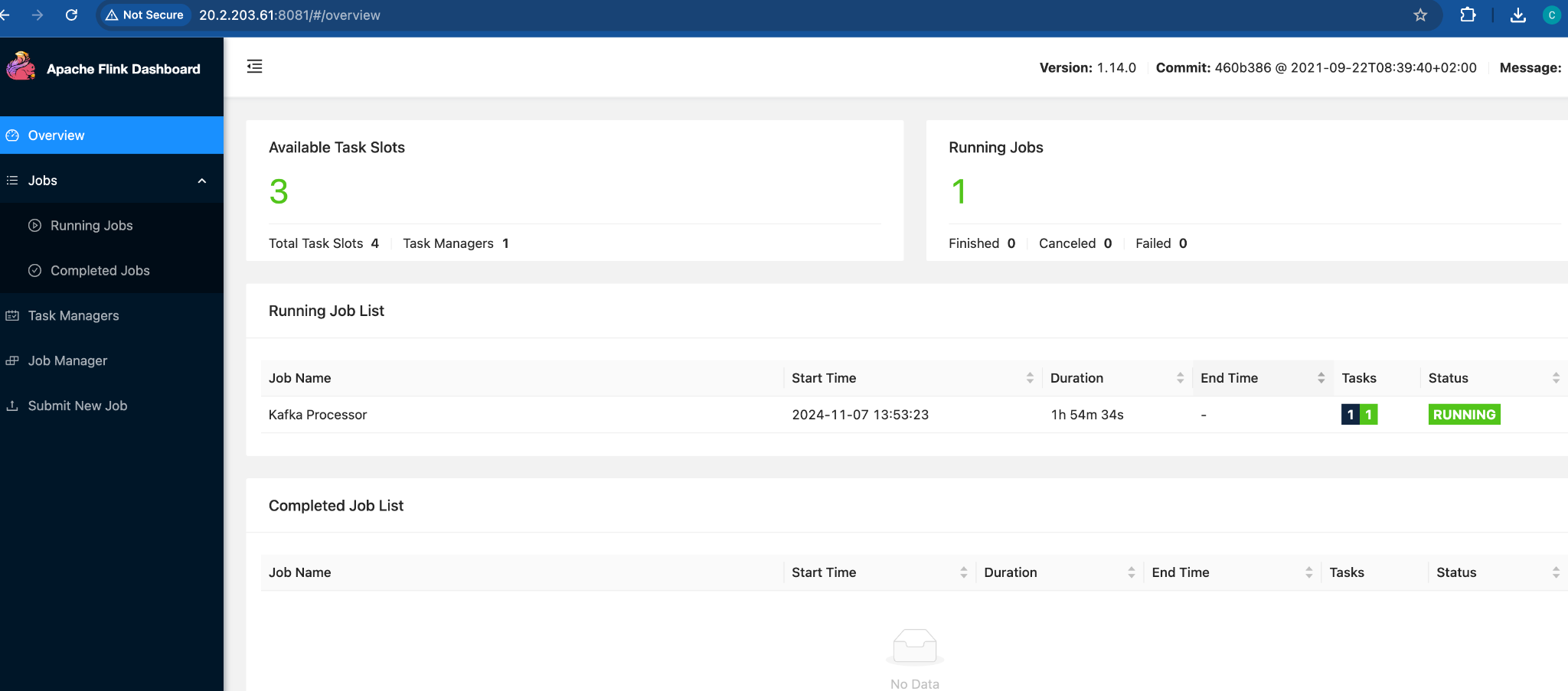
1. Screenshot showing Even and Odd output (combined view).

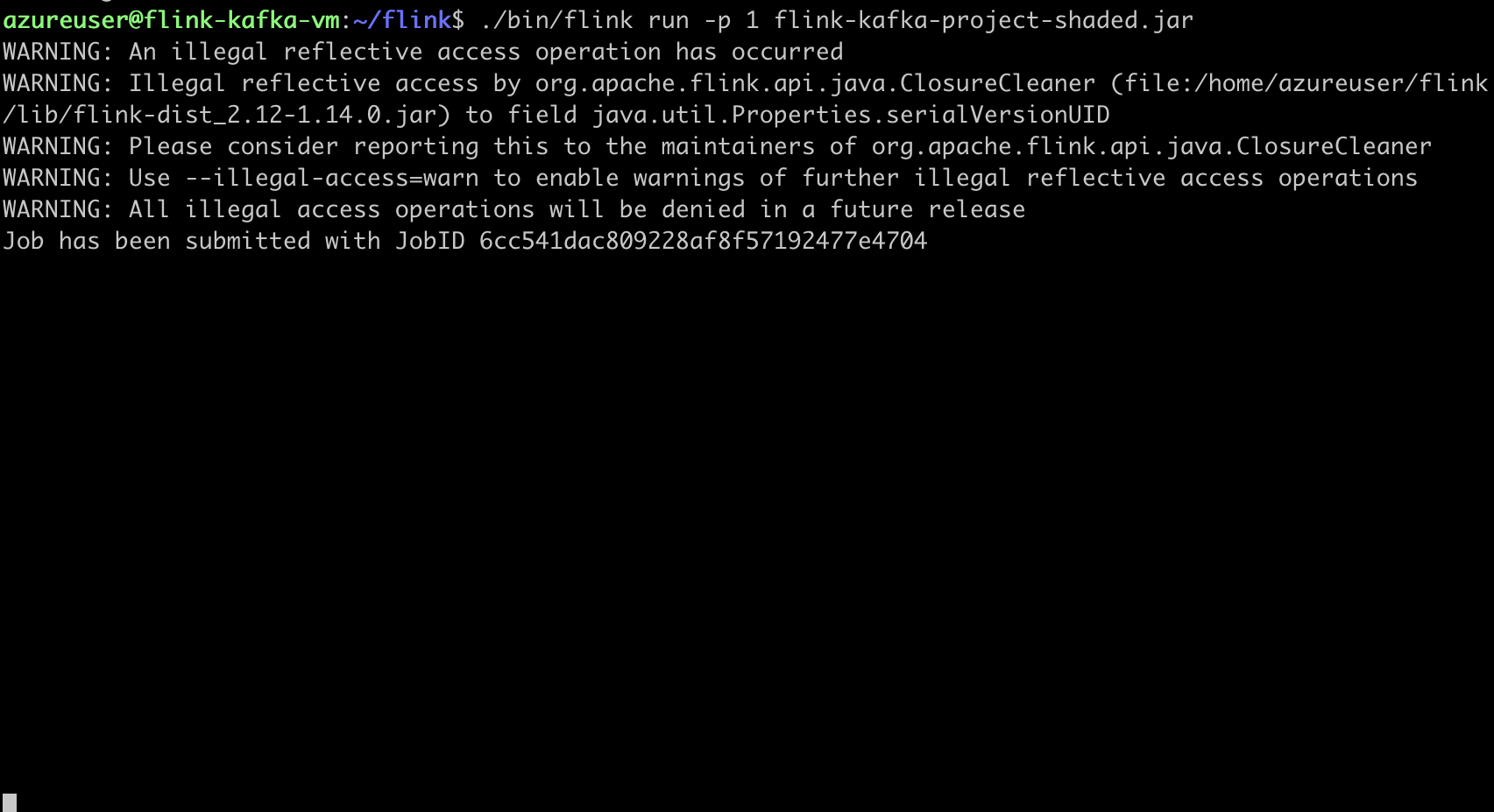


2. Screenshot of the compiled Java Maven package.



3. Screenshot of running flink application:





.